MAGATARI™

VOLUME 1 NUMBER 10

for the ATARI 400/800

1	SCHEDULE G
2	ROAD RACE
3	ATARI BASIC #1
4	6502 TUTORIAL #4
5	SOUND CONNECTOR

$14.95 for disk, $12.95 for cassette

EDITOR'S NOTES

by Andrew Hock

... sofa, stove, refrigerator, computer...

Just a few years ago, it was predicted that by the year 2000 over 75% of American homes would have a personal computer. Today, we are pushing for that prediction to come true way before that time. Unfortunately, these machines that we go out and pay so much money for are not being used to their fullest extent.

In most families, it appears that one person is stuck in front of the computer while the other members anxiously await results, whose "where-from" they do not understand. Now that your ATARI has become part of your household inventory, it should be something that the entire family can benefit from and enjoy. With MAGATAR, your ATARI can become a family treasure.

Through MAGATAR and our various software packages, we give every member of the family their turn at sitting in front of the computer, and the opportunity to enjoy it just as much as watching television.

In this issue we are introducing our BEGINNER'S CORNER column aimed at helping women as well as men, girls and boys understand, enjoy, and even come to love their ATARI computers.

We invite our readers to address any questions, no matter how trivial it might seem, to us at MAGATAR, and we welcome any comments or suggestions that you may have regarding programs or future issues of our magazine.

"MAGnify your ATARI with MAGATAR"

I INTRODUCTION TO FEATURES

1- BEGINNER'S CORNER: an introduction to the basic definitions of RAM, ROM and capacity of internal storage.

2- SCHEDULE G: Income Averaging - calculates average income for the past four (4) years for income tax purposes.

3- ROAD RACE: a car race against three other top notch drivers! A race for survival!

4- ATARI BASIC #1: lesson on ATARI GRAPHICS that explores the COLOR and SETCOLOR statements.

5- 6502 TUTORIAL #4 - GRAPHICS: explains how to fully utilize graphics on the ATARI computer.

6- SOUND CONNECTOR: our utility of the month explains how to attach your computer to your stereo equipment to get fantastic sound emissions for your games and music programs.

II - BEGINNER'S CORNER

by Gay Parris

4K?!? 16K?!? RAM?!? ROM?!? WHAT?!?

But all I want is a little computer?!?

The first thing most people realize when they decide to buy a computer is that they cannot accomplish even a simple task without first understanding another foreign language, very often referred to as "Computer Literacy". And even after getting the computer home and playing around with it a bit, you keep encountering a most frustrating message:

"OUT OF MEMORY" - better known as "ERROR 2".

Well we are about to clear up this little mystery and get you on your way to becoming "computer literate".

We can think of a computer in pretty much the same manner that we think of a brain. As long as we learn a language and have it ingrained in us, we can keep feeding new instructions and information into the brain and need not be concerned about whether or not it is "filled up". Computers can be fed information in the same manner, except that they do get "filled up".

The amount of space that you have for holding information in your computer is measured in KILOBYTES. A kilobyte (K) is 1024 bytes. A byte refers to the computer's representation of any letter, number, or special character. Thus, a computer is identified by the maximum amount of data and instructions that it can accommodate. A 16K computer holds 16,383 bytes of information.

ROM (Read-Only-Memory) is that part of the computer's

circuitry reserved for the programming language (BASIC, ASSEMBLY, etc.) which allows the computer to read instructions. ROM is not accessible to the user, so the contents of it cannot be changed. 10K ROM contains 10,240 bytes of programming language information.

RAM (Random-Access-Memory) is that area of the computer that is used for writing programs, manipulating data, and storing results. The more data and instructions a program utilizes, the more RAM is required. An ERROR 2 is really telling you that do not have enough RAM to accomplish your task. For most home applications, a minimum of 16K RAM is usually sufficient. All MAGATAR programs are written for 16K RAM. You need not restrict yourself to the amount of RAM your computer comes with; additional memory can usually be added (if you want to spend the money).

Other terms commonly used to refer to RAM are Memory, Internal Storage, Main Storage,or Primary Storage.

III - LOADING INSTRUCTIONS

PROGRAM NAMES	CASSETTE COUNTER #	DISKETTE INSTRUCTIONS
MAGATAR MENU	*NOT APPLICABLE*	RUN"D:MAGATAR"
SCHEDULE G	10	RUN"D:SCHEDULG"
ROAD RACE	97	RUN"D:RACECAR.BAS"
ATARI BASIC #1	131	RUN"D:PP.COL"
6502 TUTORIAL #4	198	RUN"D:TUTORIAL.4"

CASSETTE: All programs here at MAGATAR are copied on both sides of the tapes. If you should have problems loading a program, please try the other side.
Turn your CRT on. Advance the tape recorder to the specified line counter and press the [PLAY] button; type in CLOAD and press [RETURN] twice. If you should receive an ERROR 138 or an ERROR 140, advance the tape a counter or two past the indicated number. If you should get an ERROR 143, load the program a counter or two before the number suggested. It may require several attempts to find the exact spot on the tape before the program can actually be loaded. Please be patient! When the program loads, we recommend that you record the counter number where the program started on your cassette recorder.

DISKETTE: Turn your CRT on; insert the MAGATAR diskette in your drive and turn it on. When the busy light goes out, turn your computer on, and the programs will be loaded through the auto-boot system. If you should get a BOOT ERROR, please turn your computer off and re-start the entire process. If an ERROR 144 or a ERROR 140 should appear on your screen more than once, have your computer or your disk drive checked. If any one of these errors should persist after several trials,

please return the disk for prompt replacement.
To load the programs, separately, enter the name listed under the instructions, and press [RETURN].

SCHEDULE G - INCOME AVERAGING

Here's a program that everyone should have in their software library. Schedule G can be used in conjunction with our other tax schedules, or used by itself to help you save taxes.

If your income this year is much greater than the average of your income for the past 4 base period years (1978-1981), you may be able to pay less tax by income averaging. To see if you qualify, load the program and follow the instructions outlined below.

The first screen to appear is the main menu:

SCHEDULE G

(1) Review Base Period
Income And Adjustments

(2) Review Computation of Averageable Income

(3) Review and Calculate Computation of Tax

(4) File Maintenance Menu

(5) Return to Main Program

What is your choice?

The first thing you will want to do is to review your base period income and adjustments. To do so, select option 1 on the main menu.

After selecting option 1, the following screen will appear:

FIGURE YOUR INCOME FOR 1978 - 1981

01 Amount from 1978 line 34....._____

02 Total 1978 Exempt. *$750....._____

03 Line 1 - line 2 (M/B > 0)....._____

04 Amount from 1979 line 34....._____

05 Total 1979 Exempt. *$1000....._____

06 Line 4 - line 5 (M/B > 0)....._____

07 Amount from 1980 line 34....._____

08 Total 1980 Exempt. *$1000....._____

09 Line 7 - line 8 (M/B > 0)....._____

10 Amount from 1981 line 34....._____

11 Income earned outside USA....._____

12 Add lines 3, 6, 9, 10, 11....._____

Key [01 - 11] or [N]ext or [RETURN]

In all areas where you are to supply information, first enter the two digits corresponding to the line number, and atter the cursor appears, enter the amount, rounded to the nearest whole dollar.

If you make a mistake, and are still on that line, you can use the Back Space (Back S) key to backtrack and correct the error. Otherwise, press the line number containing the error, and re-input the correct data.

If you accidentally select a line you do not wish to use, input 0, then press RETURN.

After entering the necessary amounts, press [N] to continue to the next page.

(2) **COMPUTATION OF AVERAGEABLE INCOME**

(13) Amt. on line 12 + 30%

(14) Taxable Income 1982 line 37

(15) Penalty for Excess. Dist'n

(16) Line 14 - line 15

(17) Comm. Prop. State

(18) Line 16 - line 17

(19) Amount from line 13

(20) Line 18 - line 19

(Averageable Income)

On this screen, you need enter only three amounts. The same rules apply for entering data as in the first screen.

If your averageable income is less than or equal to $3000, you do not qualify for Income Averaging. If this is the case, you will be informed by a message on the screen, and will be returned to the main menu. If you do qualify, you will be

asked if your filing status is:

1. Single taxpayer

2. Unmarried head of household

3. Married taxpayers filing jointly, or qualifying widows and widowers

4. Married taxpayers filing a separate return

After selecting the appropriate option, the screen will automatically print out your tax computation using the income averaging method (Option 3).

Line 34 displays the tax you owe under this method. Beware, compare this amount with what you would normally have to pay because it is not always the best method, even if you do qualify.

(4) **FILE MAINTENANCE MENU**

After you have entered all the necessary data, you are ready to save your tax files. To load or save the results calculated by the program, choose option 4 from the main menu. This screen will appear:

1040 FILE LOAD

(1) Cassette Load

(2) Diskette Load

(3) Cassette Save

(4) Diskette Save

(5) Return to Main Menu

What is your choice?

Choose the appropriate option to either load or save the tax data. Remember to insert a pre-formatted diskette or cassette in your disk drive or tape recorder when saving your data. Press any key to save.

Option 5 of the 1040 File Load menu allows diskette users to return to the Schedule G Menu. Cassette users should choose this option to exit the program.

(5) **RETURN TO MAIN PROGRAM**

-- Diskette users should select this option to return to the main program menu (MAGATAR)

-- Cassette users select this option to exit from the program

GAME OF THE MONTH - RACE CAR

by Darren DeLoach

This game puts you at the wheel of your own Indy racing car - pitted against three other top racers on a winding circular track. You aren't racing against any clock - you are racing to survive!

Your score will be based on how far you go and how many cars you pass.

Your car may do one of four things:

Move left or right; or shift gear up or down. Use a joystick in port 4.

Move the stick left or right to move your car. Push the joystick to shift up a gear, and hold it there to continue up-shifting. Pull the joystick back to shift down; you may hold it there to continue down-shifting. There are 5 gears: on gear one, the other cars will pass you; gear two makes your car match the speed of the others; gears three thru five will enable you to pass the other cars.

You have three cars to start with. You lose a car by crashing into a wall or another car. Your score is given at the end of the game, and you may play again.

6502 TUTORIAL #4 - GRAPHICS

by Andrew Hock

In issue #9, we showed you how to multiply and divide using the microprocessor in your ATARI. Next issue, we'll show you how you can get your ATARI to do all the work for you. How? Well, you'd be surprised at how much your ATARI 400/800 does without using the BASIC cartridge. Many things that you think are done by BASIC commands such as '+', '-', '*', GRAPHICS, etc., actually are branched off to Operating System routines.

Which brings us to the subject that we're going to talk about in 6502 tutorial #4 - GRAPHICS. That's right, graphics are not accomplished in the BASIC cartridge. Instead, addresses are loaded and passed to a section of the OS called CIO (Central Input/Output Utility). This is where all input and output is controlled. The OS doesn't care if you're printing to the screen, the printer, the disk drive, the screen editor, or any other device (or inputting from any of the above). This feature is called device independence, and is a very handy feature of the ATARI.

For instance, executing a GRAPHICS 0 is the same as the following:

```
OPEN #6, 12+16, 0, "S:"
```

Executing a GRAPHICS 2+16 is the same as typing this:

```
OPEN #6, 12, 2+16, "S:"
```

The reason why we use IOCB (Input/Output Control Block) #6, is that this is the IOCB that is normally used by the OS for executing graphics commands (such as the above). However, we can use any IOCB from 1-7, if we wish. IOCB #0 is reserved for the Editor, and we recommend that it not be used.

So how does CIO work? Well the most important part of this utility is the IOCB (see above). There are 8 IOCB's starting at $340 (832 dec), and each one is 16 bytes long. An IOCB can access any device (ie S:, E:, D:, etc.)..... all you have to do is set up an IOCB for OPEN, then INPUT PRINT, GET or PUT to the device, and then set up the IOCB opened for CLOSE. The process is very similar to opening and closing files in BASIC.

For example, examine the two following programs:

```
(1) 10 GRAPHICS 2
    20 PRINT #6;"FUTUREHOUSE"
```

```
(2) 10 TRAP20:CLOSE #1
    20 OPEN #1,12+16,2,"S:"
    30 PRINT #1;"FUTUREHOUSE"
    40 CLOSE #1
```

These two programs do the same thing. The only difference is that program 2 uses an algorithm that is more similar to what we will do in our 6502 routine:

```
10  JSR  CLOSE
15  LDA  #$02
20  JSR  SCREEN
30  JSR  PRINT
40  JSR  CLOSE
```

Before we look at the actual routines, let's talk about some important IOCB addresses.

ICCOM COMMAND BYTE HEX $0342 DEC 834

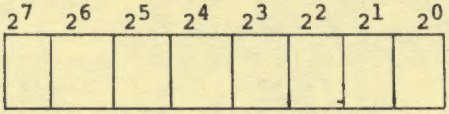
This address tells CIO what you want to do. For example, if you want to open an IOCB, the command byte is $03. Here are some other command bytes:

	HEX	DEC
CLOSE	= 0C	12
GET CHARACTERS	= 07	7
PUT CHARACTERS	= 0B	11
INPUT RECORD	= 05	5
PRINT RECORD	09	9
GET STATUS	0D	13

XIO >0D >13

ICSTA STATUS BYTE $0343 835

This byte is used to pass on the status of the result of a CIO call. The format of ICSTA is as follows:



_____ bit 7 is equal to 0 if no error has occurred
_____ if bit 7 is 1, then the remaining 7 bits indicate the error type

ICBADR BUFFER ADDRESS $0344 836
$0345 837

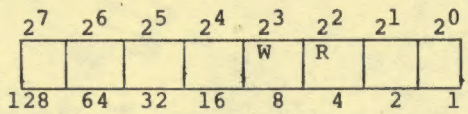
This is a two byte pointer pointing to an address which is the beginning of a buffer. The buffer can hold either data to be read or written, or a filename in the case of an OPEN command.

ICBLEN BUFFER LENGTH/ $0348 840
BYTE COUNT $0349 841

ICBLEN is a two byte address which contains the length of the buffer pointed to by ICBADR (see above).

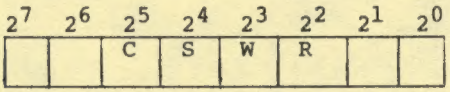
ICAUX1 AUXILLIARY INFORMATION $034A 842
ICAUX2 $034B 843

ICAU1 is used to specify whether a read or write operation is to take place. The format is as follows:



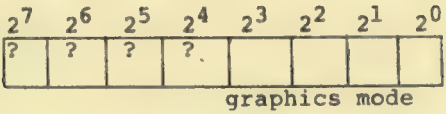
Where 'W' stands for write and 'R' represents read. The other bits of ICAX1 and all bits of ICAUX2 are used to hold device specific information. In this lesson we'll concern ourselves with using the screen handler (ie. graphics).

For graphics, ICAUX1 holds the following information:



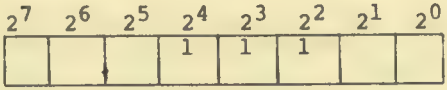
'W' and 'R' stand for read/write as above. 'S' represents split screen capabilities. If S = 1 (modes 1 thru 8) then there will be a split screen. 'C' = 1 will inhibit a screen clear on a OPEN command.

ICAUX2 contains the following:



Don't let all these addresses scare you away!
Let's take a look at line 20 in our example BASIC program

The 12+16 is the information that will be stored in ICAX1 as follows:



16 + 8 + 4
16 + 12
28

Where bit 4 specifies that a split screen will take effect, and bits 3 and 2 allow both reading and writing to the screen editor.

Now, we're ready to start using some machine language graphics. Load in our 6502 Tutorial Program, and type in RUN... So you get some graphics on the screen. Big deal, right?

The first program will load, as soon as you press a key. After it loads, you will be told:

STOPPED AT LINE 105

Type in GOTO 32000, then hit RETURN. All this does is make a USR call to our machine language routine. You will see a GRAPHICS 2 screen with a text window. Now hit any key(s), and see what happens (type in 'E' to end the subroutine).

We know what you're thinking.... Why aren't the keys I press the ones that show up on the screen? Well, the problem is that there are three types of key codes. The first is the normal ATASCII values that you're used to seeing in BASIC. The second is an internal representation of the ATASCII characters. The third, which is what you're seeing now, is the internal key codes for each character. Since we look directly at the OS vector for the key pressed, and perform no translating, the internal representation of the character is what you see.

Here's how the program works:

```
E456        10 CIO        = $E456
0342        20 ICCOM      = $342
0344        30 ICBADR     = $344
```

```
0348          40 ICBLEN      = $348
034A          50 ICAUX1      = $34A
034B          60 ICAUX2      = $34B
02FC          70 CH          = $2FC
0055          80 HORIZ       = $55
0054          90 VERTIC      = $54
0003          0100 OPENC     = 3
000C          0110 CLOSEC    = 12
0004          0120 OPIN      = 4
000B          0130 CPBINR    = 11
0002          0140 MODE      = 2
0008          0150 OPOUT     = 8
0000          0160         * = $0600
```

First let's go over some of the OS vectors and ROM routines we'll be using. $E456 is the address in ROM where the CIO utility begins. ICCOM, ICBADR, ICBLEN, ICAUX1, and ICAUX2, we've already discussed. CH ($2FC) is the OS address where a key pressed is stored before processing. HORIZ and VERTIC are two addresses that hold the current horizontal and vertical location of the cursor in any graphics mode.

The rest of the variables are commands used by ICCOM, ICAUX1 and ICAUX2. We'll explain these as we go.

```
0600 68      0170           PLA
0601 201D06  0180 SGRAPHIC  JSR SCLOSE
0604 A902    0190           LDA  #MODE
0606 202806  0200           JSR  SOPEN
0609 A200    0210 AGAIN     LDX  #$00
060B A900    0220           LDA  #$00
060D A000    0230           LDY  #$00
060F 204706  0240           JSR  SPRINT
0612 206706  0250           JSR  GETCHAR
0615 ADFC02  0260           LDA  CH
0618 C92A    0270           CMP  #$2A
061A D0ED    0280           BNE  AGAIN
061C 60      0290           RTS
```

Here's our main routine! That's right only 9 lines long. The algorithm goes like this:

1. To make sure an IOCB is 'CLOSE'd before we try to 'OPEN' it, we must close it ourselves and ignore any errors we might get.
2. Then we load the accumulator with the graphics mode we want. (MODE = 2)
3. We OPEN the IOCB.
4. We store zeros in the accumulator, X, and Y registers. These registers are used to pass the horizontal and vertical position where we want to PRINT or PLOT characters on points. When you ran the program, you probably noticed that the characters were printed at the top left corner of the screen. This is because we stored zeros in the registers here.
5. Now we are ready to PRINT or PLOT to the screen.
6. Now, we're ready to check to see if a key is pressed, and if that key is an 'E'. If it is, we return from the subroutine. If it isn't, we go back to print this

character.

```
061D A210    0300 SCLOSE LDX  #1*$10
061F A90C    0310        LDA  #CLOSEC
0621 9D4203  0320        STA  ICCOM,X
0624 2056E4  0330        JSR  CIO
0627 60      0340        RTS
```

Here's the subroutine for 'CLOSE'ing an IOCB (in this case IOCB #1). Notice that we multiply the IOCB # by $10 (16 decimal). This is because each IOCB is 16 bytes long. As with all CIO calls, we store a command (in this case CLOSEC = $0C) in the Command Byte, and pass this value to the CIO routine.

```
0010             TEXTW   =   $10
0628 A210    0350 SOPEN  LDX  #1*$10
062A A903    0360        LDA  #OPENC
062C 9D4203 0370         STA  ICCOM,X
062F A971    0380        LDA  #DEVICE&$FF
0631 9D4403 0390         STA  ICBADR,X
0634 A906    0400        LDA  #DEVICE/$100
0636 9D4503 0410         STA  ICBADR+1,X
0639 A902    0420        LDA  #MODE
063B 9D4B03 0430         STA  ICAUX2,X
063E A91C    0440        LDA  #$1C
0640 9D4A03 0450         STA  ICAUX1,X
0643 2056E4 0460         JSR  CIO
0646 60      0470        RTS
```

This is the way we OPEN a device. Again, we load the X register with the offset into the IOCB we wish to access. Then we store the number for the OPEN command in the Command Byte (ICCOM). As we stated previously, ICBADR is a two-byte address that holds the address where a file is located, or in this case, the device name to be 'OPEN'ed (S:). All we are doing here is dividing two byte address into a low byte and a high byte so we can store it. Then we store the graphics mode desired into ICAUX2, and OPIN + OPOUT + TEXT into ICAUX1. As stated before, this will allow us to input, print, and have a text window in this graphics mode.

```
063F 205806 0410 SPRINT JSR  SPOSITION
0642 A210    0420        LDX  #1*$10
0644 A90B    0430        LDA  #CPBINR
0646 9D4203 0440         STA  ICCOM,X
0649 A900    0450        LDA  #$0
064B 9D4803 0460         STA  ICBLEN,X
064E 9D4903 0470         STA  ICBLEN+1,X
0651 205F06 0480         JSR  GETCHAR
0654 2056E4 0490         JSR  CIO
0657 60      0500        RTS
```

We hope you're begining to see the similarities between OPENing, PRINTing, etc., using CIO. All we do is load appropriate addresses and pass this information to CIO. Everything else is done for us!

Next, we use CPBINR (Command for Put BINary Record) to plot or print a point or character to the screen. Then we let CIO know that the length of the buffer where our data to be printed is stored is 0, by storing zeros in ICBLEN. Next we get a character from the keyboard, and then call CIO.

```
0660 8655    0580 SPOSITION STX HORIZ
0662 8556    0590           STA HORIZ+1
0664 8454    0600           STY VERTIC
0666 60      0610           RTS
0667 ADFC02  0620 GETCHAR   LDA CH
066A C9FF    0630           CMP #$FF
066C D002    0640           BNE OUT
066E F0F7    0650           BEQ GETCHAR
0670 60      0660 OUT       RTS
0671 53      0670 DEVICE .BYTE "S:",0
0672 3A
0673 00
```

Here are the miscellaneous subroutines that help us do our work. Notice the BYTE line named DEVICE. This is the way we name our devices. It's almost as easy as BASIC!

Now that we understand how the program works, let's play around with it, First, let's change where the letters print on the screen. We can do this by changing the value at locations 1546 ($060A), and 1550 ($60E). Type in the following:

```
POKE 1546, 15: POKE 1550, 8: B=USR(1536) Then RETURN
```

Now the characters are printed towards the bottom right hand portion of the screen. Try inputting some of your own values in 1546 and 1550. If you try a value that is out of range for graphics mode 2, you will not get an error message...it simply won't print anything.

Next try changing the graphics mode. To accomplish this, all we have to do is change the value in 063A (1594 decimal). This address is in the SOPEN subroutine (see line 420). We can change this to any number from 0 to 11 decimal. Type in the following:

```
POKE 1594, 5: B=USR (1536) then RETURN
```

Try some others....well, there are some other things we can look at. For instance, we can also get rid of the text window by subtracting 16 decimal from the value in 063F, (1599 decimal). The value now is 28 which is arrived at as discussed above. Type in the following:

```
POKE 1599,(PEEK(1599) AND 16): B=USR (1536) then RETURN
```

Don't be afraid to look at the code, and make your own changes. Of course, there is the danger of locking up your ATARI when doing this, but it will only hurt your feelings.....not your computer.

When you're done fooling with this program, type in the following:

```
GOS.8100:GO TO 32000 then RETURN
```

Now we're drawing some lines. Here's an example of drawing using the CIO. The only addresses we've added are the following:

```
0011    0180  CDRAW  =  17
0001    0190  SCOLOR =  1
02FD    0195  ATACHR =  $02FD
```

CDRAW is the command for draw that must be passed to CIO via the Command Byte. SCOLOR is the color register we want to use (remember, there are four color registers).

```
0600 68      0225            PLA
0601 204206 0230 SGRAPHIC    JSR SCLOSE
0606 204D06 0250             JSR SOPEN
060E A20A    0280            LDX #$A
0610 A900    0290            LDA #$0
0612 A00A    0300            LDY #$A
0614 206C06 0310             JSR SPRINT
061A A223    0330            LDX #$23
061C A900    0340            LDA #$0
061E A00A    0350            LDY #$A
0620 208F06 0360             JSR SDRAW
0623 A223    0370            LDX #$23
0625 A900    0380            LDA #$0
0627 A023    0390            LDY #$23
0629 208F06 0400             JSR SDRAW
062C A20A    0410            LDX #$A
062E A900    0420            LDA #$0
0630 A023    0430            LDY #$23
0632 208F06 0440             JSR SDRAW
0635 A20A    0450            LDX #$A
0637 A900    0460            LDA #$0
0639 A00A    0470            LDY #$A
063B 208F06 0480             JSR SDRAW
063E 204206 0490             JSR SCLOSE
0641 60      0500            RTS
```

Here is our new main routine. If it looks long it's only because we wanted to keep it as straight forward as possible. As when plotting, we close IOCB#1, then open it, and then print giving the beginning coordinates (10,10). Then all we do is make four calls to the SDRAW subroutine, making sure that a HORIZ and VERTIC value is passed on to SDRAW also.

```
068C A501     0830
068F 208506   0850  SDRAW  JSR  SPOSITION
0692 A501     0860         LDA  SCOLOR
0694 8DFB02   0870         STA  ATACHR
0697 A210     0880         LDX  #1*$10
0699 A911     0890         LDA  #CDRAW
```

```
069B 9D4203  0900          STA  ICCOM,X
069E A90C    0910          LDA  #$0C
06A0 9D4A03  0920          STA  ICAUX1,X
06A3 A900    0930          LDA  #$00
06A5 9D4B03  0940          STA  ICAUX2,X
06A8 2056E4  0950          JSR  CIO
06AB 60      0960          RTS
06AC 53      0970  DEVICE .BYTE "S:",0
06AD 3A
06AE 00
```

The draw routine is very similar to the print routine. The only difference is that we use the CDRAW command instead of the print command (or put binary record). Also, we load ICAUX1 with a 12 (decimal) to ensure input and output. Then we make the USR call to CIO, and we're drawing!

Now let's change some addresses, and see what happens.

We can make the same changes as in the first subroutine, but now that we're drawing, let's change the picture somewhat. Type in the following:

```
POKE 1555, 90: G. 32000 then RETURN
```

Instead of a rectangle, we have a more unique looking form solid figure. Type this in:

```
POKE 1577, 160: G.32000 then RETURN
```

There are other points you can change to alter the shape. Let's change our drawing color. Type in POKE 1675, 2:G.32000 then RETURN. Our lines disappear because we're in Graphics Mode 9. Lets take a look at some other graphics modes. Type in the following:

```
GOS.8100:POKE 1623,5: G.32000 then RETURN
```

Now we have a graphics 5 screen with a text window, and our rectangle is now a square! Type in POKE 1623,0:G.32000 then RETURN.... That's right, you can draw in graphics 0 mode also. Try some other graphics modes. Change the colors, etc.

When you've done playing with the second program, type in GOS.8200:G.32000 then RETURN

Now we're filling in our rectangle. If this looks vaguely familar to the XIO command.... you're right. The only thing we added to subroutine #2 was the following:

```
0003      0220  FCOLOR  =  3
0012      0200  CFILL   =  18
02FD      0100  FILDAT  =  $2FD
```

FCOLOR is the color we want to fill the rectangle with. CFILL is the Command Byte command for fill (18 decimal).

FILDAT is the location where we must store the fill color desired. CIO looks to this address for this information when executing a 'fill'.

```
0636 A503     0480        LDA  FCOLOR
0638 8DFD02   0490        STA  FILDAT
063B A20A     0500        LDX  #$A
063D A900     0510        LDA  #$0
063F A022     0520        LDY  #$22
0641 20B206   0530        JSR  SFILL
0644 204806   0540        JSR  SCLOSE
```

In the main routine, we added the above after the last draw command, and before the close (we didn't have to do this.....XIO automatically closes the IOCB). First we store our fill color in FILDAT, then we load the register with our initial X and Y coordinates, and call the SFILL subroutine.

```
06B2 208B06 1020  SFILL  JSR  SPOSITION
06B5 A501   1030         LDA  SCOLOR
06B7 8DFB02 1040         STA  ATACHR
06BA A210   1050         LDX  #1*$10
06BC A912   1060         LDA  #CFILL
06BE 9D4203 1070         STA  ICCOM,X
06C1 A90C   1080         LDA  #$0C
06C3 9D4A03 1090         STA  ICAUX1,X
06C6 A900   1100         LDA  #$00
06C8 9D4B03 1110         STA  ICAUX2.X
06CB 2056E4 1120         JSR  CIO
06CE 60     1130         RTS
```

Here is our SFILL subroutine. It's almost exactly like the draw routine. The only difference is that we used a different command, CFILL instead of CDRAW. If you want to change the color that is used to fill the rectangle, POKE another value into 1591 decimal (0637 DEC). If you look at line 480 above, you can see that this is where the accumulator is loaded with FCOLOR.

In our next issue we're going to show you faster ways of doing graphics. Until then please let us know any comments or suggestions you have regarding the new format of our 6502 Tutorials.

UTILITY OF THE MONTH - SOUND CONNECTOR

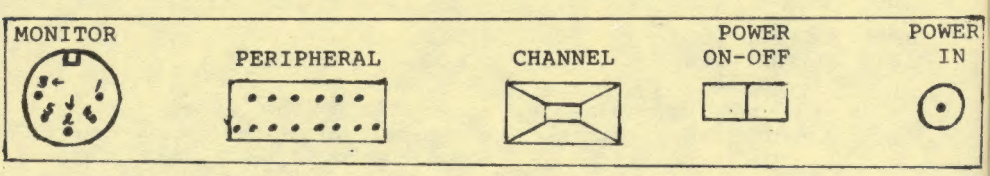
by Andy Hock

This issue our utility is not a program - it's a piece of hardware! We will explain how you can attach your ATARI to your stereo to get sound well beyond the capabilities of any television or monitor.....and your games, music programs, etc., will sound incredible!

First, let's look at where the sound is coming from - look at the right side of your ATARI. On the far right side of the control panel there is a jack marked "MONITOR". This jack is the output line to a video monitor - a stripped-down

television whose sole purpose is to provide a computer video screen - not pick up television channels! Since you are probably using your television hooked up with the cable in the back of your ATARI, you aren't using this monitor jack.

In fact, there is an "AUDIO OUT" line on this jack, in addition to the monitor out lines. Here is a diagram to show the out lines pertinent to our discussion:



* Pins 1, 4, and 5 are Video output lines, and so we can ignore them.

* Pin 2 is the "Ground" connection. We will use this pin to ground our audio line.

* Pin 3 is the audio output line. Here is where we will get our audio signal.

To make our stereo connector cable, you will need to purchase or acquire the following:

1. A five-pin male DIN Plug. Make sure the pins line up with the jack on the computer. You can buy one from Radio Shack, part numbers 274-003 for $1.49, or from APX (Atari Program Exchange), part # - APX-90002, for $2.49.

2. An audio cable with a standard Phone plug at one end and bare wires at the other. Radio Shack number 42-2371 (72 inches, $1.89) or RS# 42-2372 (144 inches, $2.19).

3. A soldering iron and some rosin-core solder.

That's all you need! Now for the real work - soldering the cables together.

What you want to do is solder the center wire of the phone cable to the pin or the DIN plug which goes into jack pin 3; and the outer wire of the phone plug to the DIN pin which goes into jack pin 2.

Remember, if you are looking straight at the DIN plug, the pin order is reversed. Pin 2 is always in the middle, but pin 3 is on the opposite side.

You will have to take the DIN plug apart to get at the soldering locations. When ready to solder, be sure you have the solder points and the bare wire hot enough to get a good joint, and then apply some solder. If you are inexperienced with soldering irons, I suggest you pick up a book or magazine which clearly explains the procedure and get a

little practice on some spare wire.

Once you have soldered the cable to the DIN plug, put it back together, plug the DIN into the monitor jack, and the phone plug into an auxiliary or tape input line on your stereo. Note that sound comes from only one channel; you must purchase a "Y-adapter, such as Radio Shack # 42-2435 ($2.59), to get the sound on both channels.

Well, that's it! Easy, right? The time is well spent if you want to enjoy high quality sound and it costs only a few dollars! Have fun!

IN ADDITION...

We would like to include programs written by our subscribers in future issues of MAGATAR, so we invite you to send us your submissions, and we will contact you for making further negotiations. All submissions should include appropriate documentation, and must be on magnetic media.

** Back issues of MAGATAR are still available at $14.95 per issue on diskette, and $12.95 on cassette plus $2.00 postage and handling. We also offer a Free Catalog of products from PROGRAMMER'S INSTITUTE.

** Any correspondence related to MAGATAR should be addressed to:

ATTENTION EDITOR
MAGATAR MAGAZINE
PROGRAMMER'S INSTITUTE
P.O. BOX 3470
CHAPEL HILL N.C. 27514
(919) 967-0861